Refine Search

Search Results -

Term	Documents
(44 AND 43).USPT.	6
(L43 AND L44).USPT.	6

Database:
US Pre-Grant Publication Full-Text Database
US Patents Full-Text Database
US OCR Full-Text Database
EPO Abstracts Database
JPO Abstracts Database
Derwent World Patents Index
IBM Technical Disclosure Bulletins

Search: L45

Refine Search

Recall Text | Clear | Interrupt

Search History

DATE: Monday, August 16, 2004 Printable Copy Create Case

Set Name side by side	Query	Hit Count	Set Name result set
DB=USPT; PLUR=YES; OP=ADJ			
L45	l43 and L44	6	L45
L44	default response	154	L44
L43	(application$ or task$1) near4 terminated	2397	L43
L42	l13 and L41	0	L42
L41	l34 and L40	18	L41
L40	wait near5 L39	214	L40
L39	user near3 response	26056	L39
L38	l36 and L37	0	L38
L37	waiting near3 user response	53	L37
L36	l13 and l34	19	L36
L35	l33 and L34	8	L35
L34	automatically sav$	712	L34
L33	l18 and user response	38	L33

L32	l20 and (priorit$ and class)	1	L32
L31	l20 and (prioti$ and class)	0	L31
L30	l20 and (prioti$ near4 class)	0	L30
L29	L20 and (amount near5 data)	0	L29
L28	L20 and (amount near5 data$1)	0	L28
L27	L20 and (recent or usage$1)	1	L27
L26	l20 and class	1	L26
L25	l20 and L24	1	L25
L24	resource$1 same (memory or capacity)	22000	L24
L23	l20 and priorit$	1	L23
L22	l20 and L21	1	L22
L21	concurrent$ or parallel$	1316890	L21
L20	5748468.pn.	1	L20
L19	l4 and L18	23	L19
L18	application$1 near4 terminated	1988	L18
L17	application$1 near4 trminated	0	L17
L16	terminated same L15	0	L16
L15	l11 near5 application$1	43	L15
L14	l11 same L13	0	L14
L13	application$ near5 terminated	2328	L13
L12	l3 same L11	1	L12
L11	lowest priorit$	4498	L11
L10	l3 and L9	1	L10
L9	l4.ab.	71	L9
L8	l5 and L7	1	L8
L7	forcibly near3 (end$ or terminat$)	2121	L7
L6	focibly near3 (end$ or terminat$)	1	L6
L5	(l2 or l3) and L4	216	L5
L4	enough near3 (resource$ or memor$)	5107	L4
L3	application$1 near2 (end$ or terminat$)	23686	L3
L2	(end$ or terminat$) near3 application$1	32046	L2
L1	(wnd$ or terminat$) near3 application$1	5558	L1

END OF SEARCH HISTORY

L7: Entry 35 of 35 File: USPT Dec 13, 1988

DOCUMENT-IDENTIFIER: US 4791556 A
TITLE: Method for operating a computer which searches for operational symbols and executes functions corresponding to the operational symbols in response to user inputted signal

Brief Summary Text (68):
Upon completing the foregoing, control is passed to a main loop which is the highest level organizational mode in the program system. The main loop is concerned with initiating transactions wherein the operator interacts with the file and/or screen, and executing the requests resulting from those transactions. The transact-execute loop is reiterated until the program is terminated by the operator or the operating system.

Brief Summary Text (98):
1. The program system of the present invention may be terminated by the operating system and therefore must be reloaded from the beginning. However, the status of the program at such termination will be saved so that the machine will be restored to the state it was prior to termination.

Detailed Description Text (40):
The information contained in the file map is also used by the program system in the preferred embodiment when a user initially loads the system, i.e., when the program system issues instructions to itself to load a file. In the current preferred embodiment whenever a user enters the system, the system will first load the file map and, based on the information stored in it, will then determine and redisplay the state of the system before prior termination. In other words, the program system will redisplay the final file viewed before the system was last terminated as specified in the file map. In the event that the system cannot locate that file, the system will so inform the user and display a default file.

Detailed Description Text (47):
The file symbol has various other related uses in the program system other than those already specified in the explanation, above. For instance, in the event that the program system in unable to locate the file which is specified by name in the second part of the file symbol, the system will automatically create and display a new file by that name for the user. Whenever a new file is thus created, the system creates a default header record and flags the file as having a temporary header record. Of course, when the file is actually saved, the default header record will be re-generated and replaced in the file and copied into the file map.

Detailed Description Text (61):
There are three general options for the present program system when a program has been called up to run: the program system could be terminated by the computer's own operating system, the program system could remain resident, or the program system could continue to run with the program being performed as a subtask. This would be dependent on the particular hardware and pre-existing operating system in use, if any, and any one or more of these options could be used, depending on the particular environment under which the present program system was being run. If the

program system were to be terminated under option 1, it would automatically save the latest status of the current file so that when the system was relocated, it would return to its exact state at prior termination, as explained above.

Detailed Description Text (69):
Thus, three basic levels of write protection for a screen would be desirable: (1.) no protection at all, i.e., the default condition, (2.) locked against any alterations, and (3.) locked against any alterations except for those areas designated by a blank field which would function like a form with fill-in-the-blank areas.

Detailed Description Text (74):
The system's default condition is provided to allow the user to TAB from symbol to symbol until he reaches the desired one, and then allows him to invoke the ENTER command to instruct the program system to display a different file, load a program, or do whatever that symbol implies to be done.

Detailed Description Text (75):
A second option is given by the system command ADVANCE, "S>ADV". Once invoked, the ADVANCE command instructs the system to advance automatically the symbol identifier to the next system symbol in the file after the system has finished carrying out the implied request of an executable system symbol on the screen; the program system simulates a manual TAB command each time a user invokes the ENTER command and the system has finished carrying out the implied request of the first system symbol. The system will then wait for the user's next input. If the user chooses to instruct the system to carry out the implied request of that next system symbol by pressing ENTER, the program system will again carry out the implied request, automatically TAB to the next system symbol on the screen, and wait for the user's next response. Alternatively, the user may choose to either TAB to some other symbol, or edit the screen. Although it is of limited use, the ADVANCE command does provide a simple means to verify that a given action designated on a screen has occurred.

Detailed Description Text (92):
It will be apparent to those skilled in the art that, as the program system of the present invention stores all interfaces created within the system in the form of a file, that a user can easily integrate any externally-created file for use in the system. Basically, the user need only designate a file symbol specifying the externally-created file's name in the second part of the symbol and invoke that symbol to bring up that file within the program system. The system will in turn, as in the case of a newly created file, generate a default header record and flag the file as possessing a temporary header record until the file is saved by the system, at which time the header record will be updated and replaced in the file map.

Previous Doc Next Doc Go to Doc#

L7: Entry 22 of 35 File: USPT Nov 30, 1999

DOCUMENT-IDENTIFIER: US 5995997 A
TITLE: Apparatus and methods for optimally allocating currently available computer resources to future task instances versus continued execution of current task instances

Brief Summary Text (27):
Furthermore, in certain situations, a future task instance(s), if precomputed, could provide greater expected value than the value of a task instance which is currently executing. In that regard, a portion of current task execution could be intentionally degraded or that task completely terminated, hence retarding or even halting execution of that task instance and freeing processing capacity, in favor of allocating that capacity to such a future task instance.

Brief Summary Text (28):
In accordance with my specific inventive teachings, the desirability of continuing execution of a currently executing task instance vis-a-vis prematurely suspending the refinement of that instance in favor of precomputing a future task instance is assessed through use of discounted net expected value (NEV) exhibited by that instance. NEV is determined as a product, of the probability of a future task instance and its EVC flux, multiplied by a suitable time-discount factor. The currently executing task instance is terminated, i.e. suspended, in favor of precomputing a future task instance if, at the onset of a time slice, the latter instance exhibits a discounted NEV that exceeds the EVC flux then being provided by the former instance. Precomputation continues for the remainder of the time slice, with a re-evaluation of discounted NEV (including that of the suspended task instance) as against the EVC flux provided by the task instance then currently executing at the beginning of each successive slice, and so forth, in order to optimally allocate currently available processing resources to their best current use.

Detailed Description Text (4):
To facilitate reader understanding, I will first discuss my invention, in detail, in the specific context of use, in an operating system, for selecting a task instance(s) for pre-computation, i.e. selecting and currently executing an instance that is likely to occur in the future and storing its results for future use--all ahead of time. I will address three different scenarios. Since these scenarios and the underlying concept of pre-computation are all independent of the particular application for which the task instances are executed or even the specific nature of the tasks themselves, for simplification, this discussion will purposely omit these latter aspects. The first two scenarios involve pre-computing, during a current interval of idle time, a task instance, from a group of instances that are likely to be executed in the future, wherein all these future task instances have: (a) fixed utility values, or (b) time-varying utility values. Thereafter, I will address the third scenario which involves prematurely terminating a task instance that is currently executing, regardless of when that task instance is executing, i.e. either during an interval of high or low processing activity (the latter including idle-time), in favor of precomputing a future task instance that possesses increased current utility over the currently executing task instance. These scenarios will be addressed first through a broad overview; which will then be followed by a discussion of the salient portions, both hardware and software, of

a computer system that embodies these three aspects of the invention.

Detailed Description Text (13):
Contrary to well-established principles of time-sharing, I have recognized that enhanced performance and throughput results if the selected task is executed to the fullest extent of the available time during this idle-time interval rather than being prematurely terminated in favor of another task. In that regard, I have found that use of a conventional time-sharing approach, under which an equal amount of time is allocated to each task instance in a group, would result in a sub-optimal allocation of available processing time.

Detailed Description Text (60):
This figure graphically depicts illustrative non-linearly varying .phi..times.p curve 410 for a currently executing task instance and illustrative linearly varying discounted .phi..times.p curve 450 associated with a future task instance pending for precomputation. Initially, assume that the task instance represented by curve 410 is executing and continues to do so through time slice .DELTA.t.sub.1. At the end of this slice, curve 410 has an illustrative magnitude l.sub.1 which exceeds a magnitude then exhibited by future task instance curve 450. Consequently, the present task instance continues to execute. At the end of the next time slice, i.e. .DELTA.t.sub.2, curve 410 still exhibits a greater magnitude, i.e. here l.sub.4, then does curve 450. Hence, current processing resources, here processing time, continue to be allocated to the presently executing task instance to further its execution and achieve current value thereby. Inasmuch as curve 410 begins to exhibit a downwardly concave shape starting in time slice .DELTA.t.sub.3 and continuing into time slice .DELTA.t.sub.4, the current task instance yields increasingly less incremental current value relative to that which can be had, discounted to the present, through precomputing the future task instance. Inasmuch as the incremental value provided by the currently executing task instance at the onset of each of time slices .DELTA.t.sub.3 and .DELTA.t.sub.4, i.e. magnitudes l.sub.4 and l.sub.3, respectively, still exceeds the EVC provided, on a discounted basis, by the future task instance at those times, processing resources continue to be allocated to the former task instance for the remainder of each of these intervals. However, at the onset of the next time slice, i.e. .DELTA.t.sub.5, the discounted EVC provided by the future task instance now exceeds the incremental value provided by current task instance, i.e. magnitude l.sub.1. Consequently, since the future task instance will provide greater EVC, discounted to the present, processing resources, here processing time, are allocated to the former rather than the latter instance. Hence, the currently executing task instance is terminated in favor of precomputing the future task instance. Inasmuch as the discounted EVC provided through precomputation of the future task instance will continue, though time slice .DELTA.t.sub.6 to exceed the EVC of the current task instance (now suspended), precomputation of the future task instance will continue through this time slice as well. A similar graphical analysis can be made between any presently executing task instance and a future task(s) instance to determine whether precomputation of the latter should occur and when.

Detailed Description Text (74):
For each task instance represented within Bayesian network 675, task value table 680 stores, in a separate entry, a measure of the value (i.e. utility, here u.sub.o (.pi.(I))) of that task instance relative to that of other such task instances for application 650.sub.1. For any given task instance, this accompanying value, as noted above, reflects an object-level value of a previously computed partial result of that instance. This value is initially defined by a programmer during program development. Update function 660 can modify this value, as needed, during application execution to reflect changes in task value over time. This value may be constant, i.e. exhibiting no variation with time, or time varying, the latter including linear or non-linear variation. This value may be represented by a function or a LUT correspondingly calculated or accessed and interpolated, if necessary, during run-time. In addition, since task value can be probabilistic in

nature and conditioned on other tasks having executed and/or other environment parameters, all such values can be collectively represented by another Bayesian network, or one integrated with network 675 to form a single unified network. In the case of non-constant probabilities or task value measures, these probabilities and task values for a common application are all reset to corresponding default values during application initialization.

Detailed Description Text (95):
Furthermore, rather than just selecting among task instances that are independent of an application or those associated with a given application, my invention could be advantageously used, in a multi-tasking environment, to prematurely invoke other applications. In that regard, while one application is executing, particularly on a personal computer, a probability exists that a user will then invoke another such application. This probability will clearly vary widely from one application to the next as well as with the current task then being executed in a current application. For example, if a word processing program is currently executing and a user, through that program, is composing a document, by, e.g., inputting text and/or graphics into a word processing file, a finite probability exists that in view of this current task, the user may well immediately invoke a web browser to obtain information from a web site for subsequent inclusion into the document. Specifically, once a user types a web address (so-called "URL") of a given site into a document, the probability associated with the user then accessing that site during the next operation may well increase. Additionally, a high probability may also exist for the user, during document composition, to re-access a site (s)he has most recently accessed during preparation of that document. As other sites are accessed or time passes without any such access, the probability associated with accessing a particular site that was previously accessed will likely decrease. Additionally, this probability will decrease if, instead of composing a document, the user is formatting a document, saving the document to a file, setting tabs in the program or executing a number of other specific tasks that are basically unrelated to entering information into a document. Inasmuch as significant intervals of idle-time occur during use of, e.g., a word processing program, by, e.g., simply waiting for keyboard or other user input, response time and overall throughput of the program can be significantly enhanced by prematurely initiating, during an idle-time interval, that program, such as a web browser, then exhibiting the highest likelihood of occurrence or .phi..times.p. In this manner, a web browser can be precomputed, i.e. in this case prematurely invoked, such that it is executing and has already accessed a site and is ready to download information at the instant a user issues an appropriate request to access that site, rather than being first invoked at that time and forcing the user to wait for the browser to initialize and then establish a connection to the desired page of the site. In essence, delay associated with program initialization and site access would occur prematurely and substantially, if not totally, during an idle-time interval, when that delay would neither be apparent to nor adversely affect a user. Of course, if the user did not issue such a request and performed a different operation, then, as time continues to pass, the likelihood or .phi..times.p associated with invoking each of these other applications would be dynamically updated to provide a new relative ranking, in terms of likelihood or .phi..times.p, respectively, of invoking other such applications. Application probabilities, given their task dependency, can be effectively represented by a Bayesian network stored within each application. During the onset of an idle-time interval, a currently executing application would interrogate its Bayesian network and select, in accordance with the algorithms discussed in detail above, an application, if any, for precomputation. The probabilities contained within this network stored within an application could be static or dynamically updated as needed to reflect the current state of that application and/or any other applications then executing as well. The associated probabilities stored within the Bayesian network would be of the following form for currently executing application i, as given by expression (15) below:

CLAIMS:

11. The method in claim 10 wherein the executing step further comprises the steps of:

if the one future task instance completes during the remaining time in said each time interval, storing results of the one future task instance for subsequent use; and

if the one future task instance does not complete during the remaining time in said each time interval, storing partial results of the one future task instance obtained at a conclusion of said each time interval and terminating the one future task instance.

Previous Doc Next Doc Go to Doc#

Lucien

L7: Entry 30 of 35 File: USPT Jul 21, 1998

DOCUMENT-IDENTIFIER: US 5784616 A
TITLE: Apparatus and methods for optimally using available computer resources for task execution during idle-time for future task instances exhibiting incremental value with computation

Brief Summary Text (27):
Furthermore, in certain situations, a future task instance(s), if precomputed, could provide greater expected value than the value of a task instance which is currently executing. In that regard, a portion of current task execution could be intentionally degraded or that task completely terminated, hence retarding or even halting execution of that task instance and freeing processing capacity, in favor of allocating that capacity to such a future task instance.

Brief Summary Text (28):
In accordance with my specific inventive teachings, the desirability of continuing execution of a currently executing task instance vis-a-vis prematurely suspending the refinement of that instance in favor of precomputing a future task instance is assessed through use of discounted net expected value (NEV) exhibited by that instance. NEV is determined as a product, of the probability of a future task instance and its EVC flux, multiplied by a suitable time-discount factor. The currently executing task instance is terminated, i.e. suspended, in favor of precomputing a future task instance if, at the onset of a time slice, the latter instance exhibits a discounted NEV that exceeds the EVC flux then being provided by the former instance. Precomputation continues for the remainder of the time slice, with a re-evaluation of discounted NEV (including that of the suspended task instance) as against the EVC flux provided by the task instance then currently executing at the beginning of each successive slice, and so forth, in order to optimally allocate currently available processing resources to its best current use.

Detailed Description Text (5):
(a) fixed utility values, or (b) time-varying utility values. Thereafter, I will address the third scenario which involves prematurely terminating a task instance that is currently executing, regardless of when that task instance is executing, i.e. either during an interval of high or low processing activity (the latter including idle-time), in favor of precomputing a future task instance that possesses increased current utility over the currently executing task instance. These scenarios will be addressed first through a broad overview; which will then be followed by a discussion of the salient portions, both hardware and software, of a computer system that embodies these three aspects of the invention.

Detailed Description Text (14):
Contrary to well-established principles of time-sharing, I have recognized that enhanced performance and throughput results if the selected task is executed to the fullest extent of the available time during this idle-time interval rather than being prematurely terminated in favor of another task. In that regard, I have found that use of a conventional time-sharing approach, under which an equal amount of time is allocated to each task instance in a group, would result in a sub-optimal allocation of available processing time.

Detailed Description Text (61):
This figure graphically depicts illustrative non-linearly varying .phi..times.p curve 410 for a currently executing task instance and illustrative linearly varying discounted .phi..times.p curve 450 associated with a future task instance pending for precomputation. Initially, assume that the task instance represented by curve 410 is executing and continues to do so through time slice .DELTA.t.sub.1. At the end of this slice, curve 410 has an illustrative magnitude 1.sub.1 which exceeds a magnitude then exhibited by future task instance curve 450. Consequently, the present task instance continues to execute. At the end of the next time slice, i.e. .DELTA.t.sub.2, curve 410 still exhibits a greater magnitude, i.e. here I.sub.4, then does curve 450. Hence, current processing resources, here processing time, continue to be allocated to the presently executing task instance to further its execution and achieve current value thereby. Inasmuch as curve 410 begins to exhibit a downwardly concave shape starting in time slice .DELTA.t.sub.3 and continuing into time slice .DELTA.t.sub.4, the current task instance yields increasingly less incremental current value relative to that which can be had, discounted to the present, through precomputing the future task instance. Inasmuch as the incremental value provided by the currently executing task instance at the onset of each of time slices .DELTA.t.sub.3 and .DELTA.t.sub.4, i.e. magnitudes 1.sub.4 and 1.sub.3, respectively, still exceeds the EVC provided, on a discounted basis, by the future task instance at those times, processing resources continue to be allocated to the former task instance for the remainder of each of these intervals. However, at the onset of the next time slice, i.e. .DELTA.t.sub.5, the discounted EVC provided by the future task instance now exceeds the incremental value provided by current task instance, i.e. magnitude 1.sub.1. Consequently, since the future task instance will provide greater EVC, discounted to the present, processing resources, here processing time, are allocated to the former rather than the latter instance. Hence, the currently executing task instance is terminated in favor of precomputing the future task instance. Inasmuch as the discounted EVC provided through precomputation of the future task instance will continue, though time slice .DELTA.t.sub.6 to exceed the EVC of the current task instance (now suspended), precomputation of the future task instance will continue through this time slice as well. A similar graphical analysis can be made between any presently executing task instance and a future task(s) instance to determine whether precomputation of the latter should occur and when.

Detailed Description Text (75):
For each task instance represented within Bayesian network 675, task value table 680 stores, in a separate entry, a measure of the value (i.e. utility, here u.sub.o (.pi.(I))) of that task instance relative to that of other such task instances for application 650.sub.1. For any given task instance, this accompanying value, as noted above, reflects an object-level value of a previously computed partial result of that instance. This value is initially defined by a programmer during program development. Update function 660 can modify this value, as needed, during application execution to reflect changes in task value over time. This value may be constant, i.e. exhibiting no variation with time, or time varying, the latter including linear or non-linear variation. This value may be represented by a function or a LUT correspondingly calculated or accessed and interpolated, if necessary, during run-time. In addition, since task value can be probabilistic in nature and conditioned on other tasks having executed and/or other environment parameters, all such values can be collectively represented by another Bayesian network, or one integrated with network 675 to form a single unified network. In the case of non-constant probabilities or task value measures, these probabilities and task values for a common application are all reset to corresponding default values during application initialization.

Detailed Description Text (96):
Furthermore, rather than just selecting among task instances that are independent of an application or those associated with a given application, my invention could be advantageously used, in a multi-tasking environment, to prematurely invoke other

applications. In that regard, while one application is executing, particularly on a personal computer, a probability exists that a user will then invoke another such application. This probability will clearly vary widely from one application to the next as well as with the current task then being executed in a current application. For example, if a word processing program is currently executing and a user, through that program, is composing a document, by, e.g., inputting text and/or graphics into a word processing file, a finite probability exists that in view of this current task, the user may well immediately invoke a web browser to obtain information from a web site for subsequent inclusion into the document. Specifically, once a user types a web address (so-called "URL") of a given site into a document, the probability associated with the user then accessing that site during the next operation may well increase. Additionally, a high probability may also exist for the user, during document composition, to re-access a site (s)he has most recently accessed during preparation of that document. As other sites are accessed or time passes without any such access, the probability associated with accessing a particular site that was previously accessed will likely decrease. Additionally, this probability will decrease if, instead of composing a document, the user is formatting a document, saving the document to a file, setting tabs in the program or executing a number of other specific tasks that are basically unrelated to entering information into a document. Inasmuch as significant intervals of idle-time occur during use of, e.g., a word processing program, by, e.g., simply waiting for keyboard or other user input, response time and overall throughput of the program can be significantly enhanced by prematurely initiating, during an idle-time interval, that program, such as a web browser, then exhibiting the highest likelihood of occurrence or .phi..times.p. In this manner, a web browser can be precomputed, i.e. in this case prematurely invoked, such that it is executing and has already accessed a site and is ready to download information at the instant a user issues an appropriate request to access that site, rather than being first invoked at that time and forcing the user to wait for the browser to initialize and then establish a connection to the desired page of the site. In essence, delay associated with program initialization and site access would occur prematurely and substantially, if not totally, during an idle-time interval, when that delay would neither be apparent to nor adversely affect a user. Of course, if the user did not issue such a request and performed a different operation, then, as time continues to pass, the likelihood or .phi..times.p associated with invoking each of these other applications would be dynamically updated to provide a new relative ranking, in terms of likelihood or .phi..times.p, respectively, of invoking other such applications. Application probabilities, given their task dependency, can be effectively represented by a Bayesian network stored within each application. During the onset of an idle-time interval, a currently executing application would interrogate its Bayesian network and select, in accordance with the algorithms discussed in detail above, an application, if any, for precomputation. The probabilities contained within this network stored within an application could be static or dynamically updated as needed to reflect the current state of that application and/or any other applications then executing as well. The associated probabilities stored within the Bayesian network would be of the following form for currently executing application i, as given by expression (15) below:

CLAIMS:

5. The method in claim 4 wherein the executing step comprises the steps of:

if the selected future task instance completes during the remaining time in the first period, storing results of the selected future task instance for subsequent use; and

if the selected future task instance does not complete during the remaining time in the first period, storing partial results of the selected future task instance obtained at a conclusion of the first period and terminating the selected future

task instance.

7. The method of claim 6 wherein, during each of said first periods, the executing step further comprises the steps of:

(a) if the selected future task instance completes during said each first period and time remains during said each first period:

1) removing the selected future task instance from the list; and

2) storing the results of the selected future task instance for subsequent use; and

3) selecting a remaining one of the future task instances in the list then having a highest product of said associated probability measure multiplied by the associated rate of change as the selected future task instance; and

(b) if the selected future task instance does not complete during the remaining time in said each first period:

1) storing partial results of the selected future task instance obtained at a conclusion of said each first period; and

2) terminating the selected future task instance;

(c) executing the selected future task instance to the extent of any time then remaining during said each first period; and

(d) repeating steps (a) through (c) above to the extent permitted by any time remaining in said each first period.

16. The method in claim 15 wherein the executing step comprises the steps of:

if the selected future task instance completes during the remaining time in said each time interval, storing results of the selected future task instance for subsequent use; and

if the selected future task instance does not complete during the remaining time in said each time interval, storing partial results of the selected future task instance obtained at a conclusion of said each time interval and terminating the selected future task instance.

18. The method of claim 17 wherein, during each of said time intervals in the group, the executing step further comprises the steps of:

(a) if the selected future task instance completes during said each time interval in the group and time remains during said each time interval in the group:

1) removing the selected future task instance from the list; and

2) storing the results of the selected future task instance for subsequent use; and

3) selecting a remaining one of the future task instances in the list then having a highest product of said associated probability multiplied by said associated rate of change (.phi..times.p) as the selected future task instance; and

(b) if the selected future task instance does not complete during the remaining time in said each time interval in the group:

1) storing partial results of the selected future task instance obtained at a conclusion of said each time interval in the group; and

2) terminating the selected future task instance;

(c) executing the selected future task instance to the extent of any time then remaining during said each time interval in the group; and

(d) repeating steps (a) through (c) above to the extent permitted by any time remaining in said each time interval in the group.

31. The apparatus in claim 30 wherein the processor, in response to the stored instructions:

if the selected future task instance completes during the remaining time in the first period, stores results of the selected future task instance for subsequent use; and

if the selected future task instance does not complete during the remaining time in the first period, stores partial results of the selected future task instance obtained at a conclusion of the first period and terminates the selected future task instance.

33. The apparatus of claim 32 wherein, during each of said first periods, the processor, in response to the stored instructions:

(a) if the selected future task instance completes during said each first period and time remains during said each first period:

1) removes the selected future task instance from the list; and

2) stores the results of the selected future task instance for subsequent use; and

3) selects a remaining one of the future task instances in the list then having a highest product of said associated probability measure multiplied by the associated rate of change as the selected future task instance; and

(b) if the selected future task instance does not complete during the remaining time in said each first period:

1) stores partial results of the selected future task instance obtained at a conclusion of said each first period; and

2) terminates the selected future task instance;

(c) executing the selected future task instance to the extent of any time then remaining during said each first period; and

(d) repeats steps (a) through (c) above to the extent permitted by any time remaining in said each first period.

42. The apparatus in claim 41 wherein the processor, in response to the stored instructions:

if the selected future task instance completes during the remaining time in said each time interval, stores results of the selected future task instance for subsequent use; and

if the selected future task instance does not complete during the remaining time in said each time interval, stores partial results of the selected future task

instance obtained at a conclusion of said each time interval and terminates the selected future task instance.

44. The apparatus of claim 43 wherein, the processor, in response to the stored instructions, during each of said time intervals in the group:

(a) if the selected future task instance completes during said each time interval in the group and time remains during said each time interval in the group:

1) removes the selected future task instance from the list; and

2) stores the results of the selected future task instance for subsequent use; and

3) selects a remaining one of the future task instances in the list then having a highest product of said associated probability multiplied by said associated rate of change (.phi..times.p) as the selected future task instance; and

(b) if the selected future task instance does not complete during the remaining time in said each time interval in the group:

1) stores partial results of the selected future task instance obtained at a conclusion of said each time interval in the group; and

2) terminates the selected future task instance;

(c) executes the selected future task instance to the extent of any time then remaining during said each time interval in the group; and

(d) repeats steps (a) through (c) above to the extent permitted by any time remaining in said each time interval in the group.

Previous Doc Next Doc Go to Doc#

Conclusion

Any inquiry concerning this communication or earlier communications from the examiner should be directed to Diem K Cao whose telephone number is (703) 305-5220. The examiner can normally be reached on Monday - Thursday, 9:00AM - 5:00PM.

If attempts to reach the examiner by telephone are unsuccessful, the examiner's supervisor, Meng-Ai An can be reached on (703) 305-9678. The fax phone number for the organization where this application or proceeding is assigned is 703-872-9306.

Information regarding the status of an application may be obtained from the Patent Application Information Retrieval (PAIR) system. Status information for published applications may be obtained from either Private PAIR or Public PAIR. Status information for unpublished applications is available through Private PAIR only. For more information about the PAIR system, see http://pair-direct.uspto.gov. Should you have questions on access to the Private PAIR system, contact the Electronic Business Center (EBC) at 866-217-9197 (toll-free).

Any response to this action should be mailed to:
Commissioner for Patents
PO Box 1450
Alexandria, VA 22313-1450

Diem Cao

Refine Search

Search Results -

Term	Documents
(17 AND 6).USPT.	105
(L6 AND L17).USPT.	105

Database:
US Pre-Grant Publication Full-Text Database
US Patents Full-Text Database
US OCR Full-Text Database
EPO Abstracts Database
JPO Abstracts Database
Derwent World Patents Index
IBM Technical Disclosure Bulletins

Search: L18

Refine Search | Recall Text | Clear | Interrupt

Search History

DATE: Monday, August 16, 2004 Printable Copy Create Case

Set Name side by side	Query	Hit Count	Set Name result set
DB=USPT; PLUR=YES; OP=ADJ			
L18	l6 and L17	105	L18
L17	l10 and L16	141	L17
L16	l1 and L15	778	L16
L15	resource$1 near3 availab$	12159	L15
L14	l3 and l12	1	L14
L13	l2 and L12	0	L13
L12	l1 and l10	262	L12
L11	l7 and L10	0	L11
L10	free$ near3 resource$1	2045	L10
L9	l7 and L8	0	L9
L8	default command	199	L8
L7	l5 and L6	35	L7
L6	default$	49499	L6

L5	l1 and l2	51	L5
L4	l1 and l2 and L3	0	L4
L3	default response	154	L3
L2	wait$ near5 (user response or (response near3 user$1))	322	L2
L1	terminat$ near3 (applicat$ or task$ or program$)	14212	L1

END OF SEARCH HISTORY